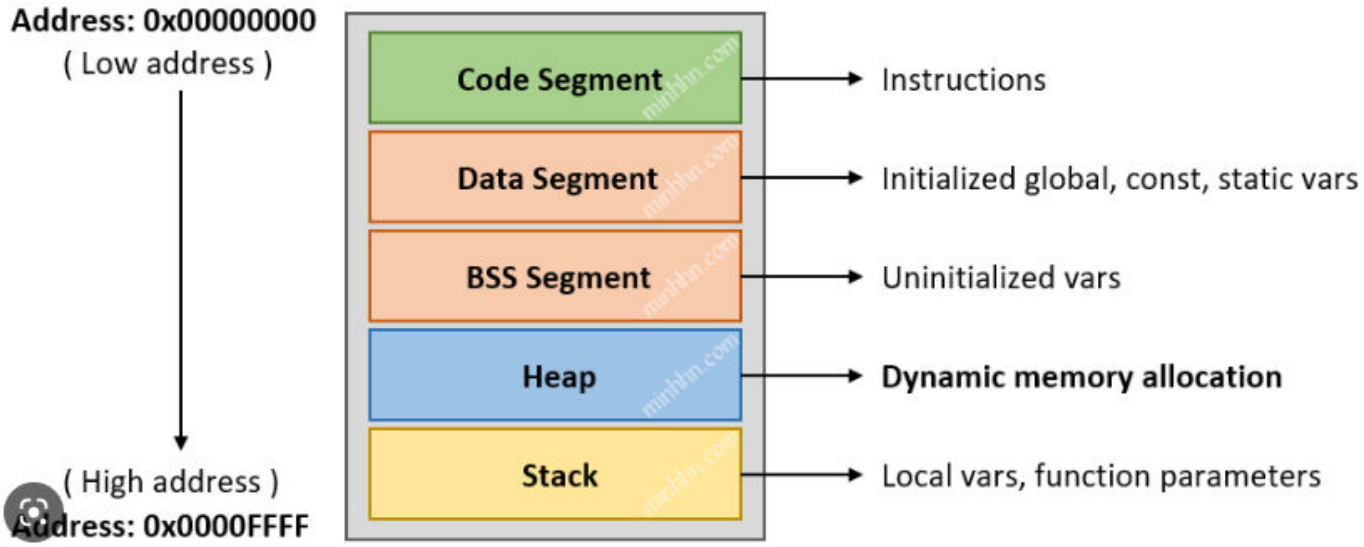
I. Phân vùng nhớ trong C

1. Phân vùng nhớ

- Gồm 5 phân vùng nhớ : Stack, Heap, BSS, DS, Text



- Text(Code segment)

+ Tất cả các mã máy trong chương trình sẽ được load vào vùng này

+ Là vùng chứa lệnh để thực thi trong chương trình

+ Chỉ có quyền truy cập Read để tránh sửa đổi

- DS(Initialized data segment)

+ Có quyền truy cập read-write

+ Vùng chứa biến toàn cục hoặc static với giá trị khởi tạo khác 0

+ Được giải phóng bộ nhớ khi kết thúc chương trình

+ VD :

int x = 1;

static int y = 2;

- BSS(Uninitialized data segment)

+ Có quyền truy cập read-write

+ Vùng chứa biến toàn cục hoặc static với giá trị khởi tạo bằng không hoặc không khởi tạo

+ Được giải phóng bộ nhớ khi kết thúc chương trình

+ VD :

int x;

stactic int y = 0;

- Heap(Dynamic Memory Allocation)

+ Có quyền truy cập read-write

+ Được sử dụng để cấp phát bộ nhớ động như Malloc, Calloc...
+ Sẽ được giải phóng khi gọi hàm free

- Stack(Automnatic variable storage)
+ Có cấu trúc LIFO(Last in, first out) Strucure
+ Có quyền truy cập read-write
+ Được sử dụng cấp phát cho biến local, input parameter ...
+ Sẽ được giải phóng khi ra khỏi block code/hàm

2. So sánh
- Sự khác biệt giữa stack và heap
+ Giống nhau : đều là vùng nhớ được lưu trữ trong RAM khi chương trình được thực thi
+ Khác nhau :
+) Stack :
. Lưu trữ các biến cục bộ trong hàm, tham số truyền vào...
. Truy cập vào bộ nhớ nhanh khi được thực hiện chương trình
. Vùng nhớ được quản lý bởi hệ điều hành, sẽ được giải phóng khi kết thúc chương trình
+) Head
. Lưu trữ vùng nhớ cho các biến con trỏ được cấp phát động bởi các hàm malloc, calloc..
. Vùng nhớ chỉ được giải phóng khi được gọi hàm free, nếu không bộ nhớ sẽ liên tục được tăng lên và có nguy cơ bị tràn

- Sự khác biệt giữa DS và BSS
+ Giống nhau
+) Đều có quyền truy cập read-write
+) Được giải phóng khi kết thúc chương trình
+ Khác nhau :
+) DS : Là vùng chưa biến toàn cục hoặc static với giá trị khởi tạo khác 0
+) BSS : Là vùng chứa biến toàn cục hoặc static với giá trị khởi tạo bằng không hoặc không khởi tạo

II. Macro and Funcion

1. Macro
- Được xử lý bởi preprocessor
- Thay thế đoạn code được khai báo Macro vào bất kì chỗ nào xuất hiện Macro đó
- Được khai báo bằng từ khóa **#define**
- Được ứng dụng để định nghĩa các biến const, hàm …
- VD : #define SUM(a,b) (a+b)

2. Funcion

- Là khối lệnh thực hiện một chức năng nào đó trong chương trình
- Được xử lý bằng compiler khi hàm được gọi
- Khi hàm được gọi, compiler sẽ phải lưu con trỏ chương trình PC(Program count) hiện tại vào Stack pointer,
- Chuyển PC tới hàm được gọi, thực hiện hàm đó xong rồi lấy kết quả trả về sau đó quay lại vị trí ban đầu trong Stack trước khi gọi hàm và tiếp tục thực hiện chương trình

3. So sánh Macro và Funcion

- Macro :

+ Chỉ thay thế đoạn code Macro vào chỗ được gọi trước khi được biên dịch
+ Thời gian chạy nhanh nhưng khiến code dài hơn bình thường
+ Không xác nhận được kiểu dữ liệu của tham số hoặc giá trị nhận được
+ Không thể debug tìm lỗi trong khi chương trình được chạy

- Funcion :

+ Phải tạo một Funcion call, lưu địa chỉ trước khi gọi hàm
+ Do phải gọi Funcion call nên tốn thời gian nhưng code sẽ ngắn gọn hơn
+ Phải xác nhận được kiểu dữ liệu của tham số hoặc giá trị nhận được
+ Có thể debug để tìm ra lỗi trong quá trình thực thi

III.So sánh giữa Struct and Union

- Giống nhau : có cách sử dụng và ý nghĩa giống nhau
- Khác nhau : khác nhau về mặt lưu trong dữ liệu trong bộ nhớ

+ Struct :

+) Dữ liệu các thành viên của struct được lưu trữ ở những vùng nhớ khác nhau
+) Kích thước của 1 struct tối thiểu bằng kích thước của các thành viên cộng lại và phụ thuộc vào bộ nhớ đệm (struct padding)

+ Union :

+) Dữ liệu các thành viên sẽ chung 1 vùng nhớ
+ Kích thước của Union sẽ được tính bằng kích thước lớn nhất của kiểu dữ liệu trong Union.
+ Việc thay đổi nội dung của 1 thành viên sẽ dẫn đến thay đổi nội dung các thành viên khác

IV. Static

1. Định nghĩa :

- Được khai báo bằng từ khóa : ***Static + kiểu dữ liệu***
- Có phạm vi truy cập trong một file,hàm , không thể truy cập từ file,hàm khác

2. Biến Stactic cục bộ

- Được khởi tạo 1 lần duy nhất và tồn tại trong thời gian chạy chương trình
- Giá trị không bị mất đi ngay ca khi kết thúc hàm, và mỗi lần hàm được gọi thì giá trị của biến chính là giá trị lần gần nhất được gọi
- Chỉ có thể được gọi nội bộ trong hàm khởi tạo ra nó

3. Biến static toàn cục

- Chỉ có thể truy cập và sử dụng trong file được khởi tạo nó, các file khác không thể truy cập

V. Con trỏ

1. Con trỏ (Pointer)

- Được khai báo bằng cách : kiểu dữ liệu + '*'
- Là những biến được lữu trữ địa chỉ bộ nhớ của những biến khác
- Do là biến được lưu dữ địa chỉ bộ nhớ nên tất cả các con trỏ đều có kích thước bằng nhau và phụ thuộc vào kiến trúc của VXL

2. Con trỏ hàm

- Kiểu khai báo : type + (*ptr)(type,…)
- Là con trỏ lưu trữ địa chỉ của 1 hàm trong bộ nhớ máy
- Được sử dụng để truy cập vào vị trí của hàm trong bộ nhớ cũng như thực thi các lệnh của hàm
- Kiểu dữ liệu khai báo và kiểu dữ liệu trong hàm con trỏ phải tương được
- VD : int (*ptr)(int x)

3. Con trỏ void

- Kiểu khai báo : void *ptr
- Là con trỏ đặc biệt, khi khai báo sẽ không cần xác định kiểu dữ liệu
- Có thể lưu trự địa chỉ của bất kì biến nào (với nhiều kiểu dữ liệu khác nhau) trong chương trình
- Vì không xác nhận kiểu dữ liệu nên khi sử dụng dể đọc hoặc ghi thì cần phải ép kiểu cho cùng loại dữ liệu muốn đọc hoặc ghi
- VD :

```
int value = 5;
void *ptr = &value;
uint32_t *ptr2 = (uint32_t *)(*ptr)
```

4. Con trỏ Null

- Là con trỏ không trỏ tới địa chỉ nào cả và có giá trị là 0
- Ứng dụng : khi khai báo con trỏ mà không biết gán giá trị nào thì ta gán vào giá trị NULL

- VD : int *ptr = NULL;

5. Mảng con trỏ

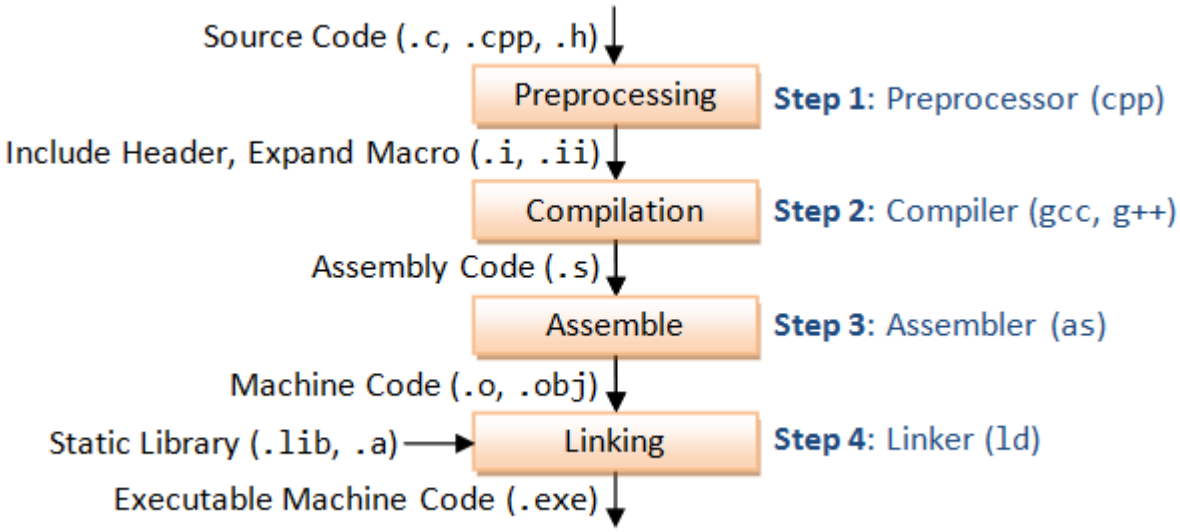
- Khai báo : type *tên mảng[]
- Là mảng mà mỗi phần tử đều là một con trỏ và có địa chỉ khác nhau
- VD : int *arr[] = { 1,2,3}

6. Pointer to Pointer

- Kiểu khai báo : **ptr
- Là loại con trỏ dùng để lưu trữ địa chỉ của con trỏ khác hay giá trị của con trỏ **ptr chính là địa chỉ của *ptr
- VD : int **ptr1 = &ptr2

VI. Compiler

1. Compiler :
- Quy trình dịch là quá trình chuyển đổi từ ngôn ngữ bậc cao (NNBC) (C/C++, Pascal, Java, C#…) sang ngôn ngữ đích (ngôn ngữ máy) để máy tính có thể hiểu và thực thi. Ngôn ngữ lập trình C là một ngôn ngữ dạng biên dịch. Chương trình được viết bằng C muốn chạy được trên máy tính phải trải qua một quá trình biên dịch để chuyển đổi từ dạng mã nguồn sang chương trình dạng mã thực thi. Quá trình được chia ra làm 4 giai đoạn chính:
+ Giai đoàn tiền xử lý (Pre-processor)
+ Giai đoạn dịch NNBC sang Asembly (Compiler)
+ Giai đoạn dịch asembly sang ngôn ngữ máy (Asember)
+ Giai đoạn liên kết (Linker)
- Sơ đồ :



2. Các giai đoạn của Compiler
- Giai đoạn tiền xử lý Preprocessor
Giai đoạn này sẽ thực hiện:
+ Nhận mã nguồn
+ Xóa bỏ tất cả chú thích, comments của chương trình
+ Chỉ thị tiền xử lý (bắt đầu bằng #) cũng được xử lý

+ *Ví dụ*: chỉ thị #include cho phép ghép thêm mã chương trình của một tệp tiêu để vào mã nguồn cần dịch. Các hằng số được định nghĩa bằng #define sẽ được thay thế bằng giá trị cụ thể tại mỗi nơi sử dụng trong chương trình.

2. Công đoạn dịch Ngôn Ngữ Bậc Cao sang Assembly
+ Phân tích cú pháp (syntax) của mã nguồn NNBC
+ Chuyển chúng sang dạng mã Assembly là một ngôn ngữ bậc thấp (hợp ngữ) gần với tập lệnh của bộ vi xử lý.

3. Công đoạn dịch Assembly
+ Dich chương trình => Sang mã máy 0 và 1
+ Một tệp mã máy (.obj) sinh ra trong hệ thống sau đó.

4. Giai đoạn Linker
+ Trong giai đoạn này mã máy của một chương trình dịch từ nhiều nguồn (file .c hoặc file thư viện .lib) được liên kết lại với nhau để tạo thành chương trình đích duy nhất
+ Mã máy của các hàm thư viện gọi trong chương trình cũng được đưa vào chương trình cuối trong giai đoạn này.
+ Chính vì vậy mà các lỗi liên quan đến việc gọi hàm hay sử dụng biến tổng thể mà không tồn tại sẽ bị phát hiện. Kể cả lỗi viết chương trình chính không có hàm main() cũng được phát hiện trong liên kết.
+ Kết thúc quá trình tất cả các đối tượng được liên kết lại với nhau thành một chương trình có thể thực thi được (executable hay .exe) thống nhất.